i-NVMM: A Secure Non-Volatile Main Memory System with Incremental Encryption*

Siddhartha Chhabra†
Dept. of Electrical and Computer Engineering
North Carolina State University
schhabr@ncsu.edu

Yan Solihin
Dept. of Electrical and Computer Engineering
North Carolina State University
solihin@ncsu.edu

ABSTRACT

Emerging technologies for building non-volatile main memory (NVMM) systems suffer from a security vulnerability where information lingers on long after the system is powered down, enabling an attacker with physical access to the system to extract sensitive information off the memory. The goal of this study is to find a solution for such a security vulnerability. We introduce i-NVMM, a data privacy protection scheme for NVMM, where the main memory is encrypted incrementally, i.e. different data in the main memory is encrypted at different times depending on whether the data is predicted to still be useful to the processor. The motivation behind incremental encryption is the observation that the working set of an application is much smaller than its resident set. By identifying the working set and encrypting remaining part of the resident set, i-NVMM can keep the majority of the main memory encrypted at all times without penalizing performance by much. Our experiments demonstrate promising results. i-NVMM keeps 78% of the main memory encrypted across SPEC2006 benchmarks, yet only incurs 3.7% execution time overhead, and has a negligible impact on the write endurance of NVMM, all achieved with a relatively simple hardware support in the memory module.

Categories and Subject Descriptors

B.3.m [**Hardware**]: Memory Structures—*Miscellaneous*

General Terms

Security

*This work was supported in part by NSF Award CCF-0915501

†Now works at Intel Labs (siddhartha.chhabra@intel.com)

ISCA'11, June 4–8, 2011, San Jose, California, USA.

Keywords

Security, Privacy, Hardware Attacks, Incremental Encryption, Non-Volatile Main Memory

1. INTRODUCTION

The trends toward increasing the number of cores on a chip in most microprocessors and increasing software complexity have put a tremendous pressure on the capacity of the main memory system. Traditional memory technologies like DRAM face serious challenges in terms of cost, energy consumption, and scalability [1, 10]. Researchers are actively researching into alternative memory technologies, and have proposed several promising candidate alternative memory technologies such as Phase-Change Memory (PCM) and MRAM. PCM seems promising because it is claimed to be $4\times$ denser than DRAM, is only $2-4\times$ slower than DRAM, and has been demonstrated in prototypes to scale better than DRAM [1, 17].

All alternatives to DRAM memory currently considered are non-volatile, meaning that they retain information even when powered off. A non-volatile main memory (NVMM) has both advantages and disadvantages over its volatile counterpart. Among the advantages of NVMM are that no refresh power is consumed to maintain code and data, and resumption from sleep or hibernation can be made instantaneous. However, there are also significant disadvantages. One disadvantage is that a typical NVMM has limited write endurance. Many studies have explored techniques to improve the write endurance of NVMM, in particular PCM [3, 9, 14, 16]. Another significant disadvantage is a *security vulnerability* where data persists on the main memory even when the system is powered down (shutdown or hibernation). An attacker with physical access to the system can simply scan the main memory content and extract all valuable information off the main memory, for example, user passwords, credit card numbers and other security sensitive data. With computing extensively becoming mobile (e.g. PDAs, laptops), and mobile devices being easily lost/stolen, attackers have more opportunities in getting physical access to systems to launch such physical attacks. NVMM's higher density compared to DRAM, coupled with the increasing need for memory capacity, make it likely for future systems using NVMM to be provisioned with much larger main memories. This implies an even larger amount of data persisting on the main memory, and increasing the incentive to attack the system.

The goal of this study is to find a solution to the security

vulnerability of lingering data in an NVMM. A satisfactory solution to achieve the goal must satisfy at least four requirements. One requirement is that we must preserve the instant-on benefit of NVMM, which means that data should be retained in memory so that it can be recovered when the processor boots or wakes up from hibernation. This requirement precludes flushing out data from the NVMM when the system is powered off. Instead, data must be retained, but in an unintelligible (i.e. encrypted) form so that attackers cannot recover any useful information from the NVMM. A second requirement is that the solution must be self contained in the memory system because its encryption ability should not depend on a particular processor platform, instruction set architecture (ISA), or require specific changes to the processor architecture. Since NVMM is targeted for a high-volume commodity memory market, an encrypted NVMM should work for a wide range of processor platforms it is attached to (servers, laptops, mobile phones, embedded systems, etc.), and its effectiveness should not be predicated on specific ISA or processor architecture changes. This requirement precludes the use of secure processor technology that requires processor-side engine to encrypt the main memory [4, 19, 20, 22] and necessitates the solution to have a memory-side cryptographic engine embedded in the memory module itself. Placing the cryptographic engine on the memory module, instead of the memory controller, avoids requiring changes to the processor chip as the memory controller can be integrated with the processor chip. The third requirement is that encrypted NVMM should be as secure as its volatile predecessor (DRAM). This places DRAM's retention time as a limit for how long it takes to complete memory encryption after power down. Thus, it is preferable to keep much of the memory encrypted at all time, and only encrypt a small amount of data upon power events. The final requirement is that the solution should not incur substantial performance or energy overheads for applications running on the system.

Contributions: In this paper, we propose *i-NVMM*, a data privacy protection scheme for non-volatile main memory (NVMM). i-NVMM relies on a memory-side encryption engine to support encrypted main memory, hence it does not rely on specific architecture support at the processor level. The "i" in "i-NVMM" stands for incremental, meaning that the main memory is encrypted incrementally, i.e. different data in the main memory is encrypted at different time depending on whether the data is predicted to still be useful to the processor. The motivation behind incremental encryption is the observation that the working set of an application (consisting of memory pages that an application is actively using) is much smaller than its resident set (consisting of all memory pages belonging to the application). By identifying the working set and encrypting the remaining part of the resident set, i-NVMM can keep the majority of the main memory encrypted at all time without penalizing application performance by much. i-NVMM employs a predictor to predict "*inert*" pages by scanning pages in the main memory periodically to identify memory pages that have not been used for a long time. Once inert pages are identified, they are sent to a memory-side cryptographic engine located in the memory module to be encrypted. Because i-NVMM identifies inert pages and encrypts them early without waiting for a power event, the *vulnerability window* at power down, the time period in which attacks can be carried out when a system is powered down, is narrowed significantly. In addition, even when powered, there is an additional security protection because much of the main memory is encrypted at all time.

i-NVMM incurs performance overhead when the processor accesses a page that is already encrypted, hence incurring page decryption latency. On such a "misprediction" event, we can choose to decrypt only the block that is accessed without decrypting the page. Alternatively, we can decrypt the entire page to prevent future accesses from paying the decryption latency. Finally, we can trigger page decryption after a number of accesses have occurred, to ensure that the page has truly become a part of the working set of the application. We experimented with different choices and examined their impact on performance overhead and encryption coverage. Another way to hide the decryption latency is to predict proactively which page is likely to be accessed in the future and pre-decrypt it ahead of time. We explore such a pre-decryption technique using page address correlation information as input to the predictor.

In order to simulate i-NVMM, we implemented a detailed cycle-accurate simulation built on top of Simics [11]. Our results show that using incremental encryption, on average i-NVMM keeps 78% of the main memory encrypted across SPEC2006 benchmarks, yet only incurs 3.7% execution time overhead, and has a negligible impact on the write endurance of NVMM, all achieved with a relatively simple hardware support in the memory module. On average, i-NVMM requires roughly 5 seconds to fully encrypt and secure a 32GB main memory at power down, matching DRAM's retention time. i-NVMM outperforms all straightforward alternative approaches. It outperforms a scheme that keeps the main memory encrypted at all time: execution time overheads of 3.7% vs. 41.5%. It also outperforms a scheme that only encrypts the entire main memory at power down: better security protection, and a reduction of vulnerability window from 23 seconds to just 5 seconds.

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 presents our assumed attack model. Section 4 presents the rationale behind the design for the proposed security system. Section 5 describes the design of i-NVMM. Section 6 shows our experimental setup and section 7 presents our results. We finally conclude in section 8.

2. RELATED WORK

In contrast to hard drive encryption, where (even software) encryption latency is a small part of disk access latency, hardware encryption latency can be as much as 50% of the main memory access latency, hence more sophisticated schemes are needed to maintain low performance overheads.

Much of the work on non-volatile main memory (NVMM) has centered around Phase Change Memory (PCM) and MagnetoResistive RAM (MRAM). Related work on PCM as NVMM has concentrated on four primary areas: bridging the latency gap between DRAM and PCM [3, 9, 15, 16], bridging the energy gap of read and especially write operations between DRAM and PCM [3, 9, 24, 26], increasing write endurance [3, 9, 14, 16], and addressing security concerns in PCM.

An area of research in NVMM relevant to our work is the one that deals with security concerns in PCM, where its limited write endurance may be exploited by an attacker to run an application that causes damage to the memory through

repeated writes [2, 14]. Qureshi et al. [14] propose randomized start-gap wear leveling algorithm which will move a line in the physical memory before it reaches its endurance limit. Zhang et al. [25] also address this potential attack by using a DRAM buffer for allocating highly modified pages to avoid excessive writes to the PCM. In addition to write endurance, another security vulnerability of NVMM is data lingering in plaintext form in the main memory long after the system is powered off, a form of *data remanence* vulnerability. To the best of our knowledge, *such a vulnerability has not been addressed with a memory-side solution.*

Many main memory encryption techniques have been proposed, among them are [4, 8, 19, 20, 22]. They involve a cryptographic engine in the processor chip and ISA support to encrypt the main memory and provide its integrity verification. While such techniques can be used to address the security vulnerability that our paper is targeting (in addition to targeting additional attacks such as bus snooping and tampering), there are merits to having a solution that is self-contained in the memory system. With a solution that is self-contained in the memory system, the encryption ability is not a function of a particular processor platform, instruction set architecture (ISA), and requires no specific changes to the processor architecture. Deploying an encrypted main memory is not possible currently because no commodity processors come with memory encryption and integrity verification support. Even if secure processors become available in the future, they may be available in only a narrow range of platforms, and may be considered a solution overkill in others. In contrast, a solution that is self-contained in the memory system can be deployed right now, and on a wide range of processor platforms it is attached to, such as servers, laptops, mobile phones, embedded systems, etc. In addition, for the memory makers, providing a self-contained secure nonvolatile main memory provides a way to differentiate their products from lower-end commodity memories. These are the key reasons why we require our i-NVMM mechanism to be self-contained in the memory system, without requiring a particular processor platform, instruction set architecture (ISA), or specific changes to the processor architecture.

3. ATTACK MODEL AND SECURITY PROTECTION

3.1 Goal of Security Protection

The basic attack that we consider in this paper is one in which an attacker obtains physical access to a system, and extracts sensitive information from the storage system by reading it. The attacker may obtain physical access to the system through various means, such as theft, or acquisition of carelessly disposed systems. Such attacks have been demonstrated to be very easy to perform on non-volatile storage. In 2003, two MIT graduate students acquired 158 hard disk drives from eBay auctions, and found that 74% of drives contained old data that could be recovered and read, and only 9% had been properly sanitized prior to disposal. Among the sensitive information retrieved from the disk drives were detailed personal and corporate financial records, medical records, etc. [18]. Recognizing such vulnerabilities, many encrypted file systems or file utilities have been produced.

All current computing systems store information in the main memory in plaintext form. There is no software solution that encrypts data in the main memory because software itself must store its code, data, and program variables in the main memory. The lack of secure main memory is acceptable to many users when DRAM main memory is used, because once powered off, information is not retained in the main memory. Unfortunately, compared to traditional DRAM main memory, NVMM incurs a new security vulnerability because information lingers on in the main memory long after the system is powered down. Thus, to address this "data remanence" vulnerability, a solution must provide the security level comparable to DRAM.

The security vulnerability of DRAM memory is strictly limited by its information retention time, i.e. how long information is retained in DRAM cells. It is well known that DRAM's retention time varies across cells, pages, and banks. Some cells retain information much longer than others. A recent study by Venkatesan et al. [23] reported that on a synchronous DRAM (16MB, IS42S16800A SDRAM manufactured by ISSI) at room temperature (24C), page retention time varies from 500 ms to 50 seconds. Close to all (99% of pages) retain information for up to 3.1 seconds, and roughly 90% of pages retain information for up to 32 seconds. Retention time declines as the environment temperature increases. At 45C, 90% of pages retain information for up to 18 seconds, while at 70C, 90% of pages retain information for up to only 3.2 seconds. In contrast, even at 118C, PCM retains data for roughly 10 years [21]. Overall, therefore, in order to match the security of DRAM operating under a range of temperature, a *secure NVMM must not retain data for more than a few seconds after the system is powered down.* [1]

The attack model considered in this paper assumes that the system may be lost or stolen in a power state (most common scenario). The attacker does not have the capability to run rogue applications to read the memory, but can read the memory physically, possibly without powering up the system. We do not attempt to protect against bus snooping or tampering of the bus or other off-chip components.

3.2 Protection from Various Approaches

Recall that our goal of protection is to match the inherent security protection from using volatile main memory (DRAM). In order to achieve that, we will employ memory encryption. There are two ways to encrypt the main memory. One approach is to encrypt the entire memory at power down (*bulk encryption*). Another approach is to keep most of the memory encrypted at all time, so that only a small percentage of pages need to be encrypted on power down. We refer to such an approach as *incremental encryption.*

Figure 1 illustrates the differences between these protection approaches in relation to what types of attacks they protect against. The figure shows an example where the processor's computation busy time occurs early and just before power down. For DRAM main memory, when the system is powered down, there is a brief vulnerability window (VW) in which the main memory still retains information. Once we pass the DRAM retention time, no information lingers and the main memory is protected from information theft.

With bulk encryption on NVMM, encrypting the entire main memory will take a long time (e.g., tens of seconds – more discussion in Section 7.3), hence the vulnerability

[1]DRAM, while non-volatile, may still suffer from a data remanence attack. For example, spraying duster spray on laptops' DRAM modules brought DRAM temperature to -50C, causing its retention time to blow up to tens of minutes [5].

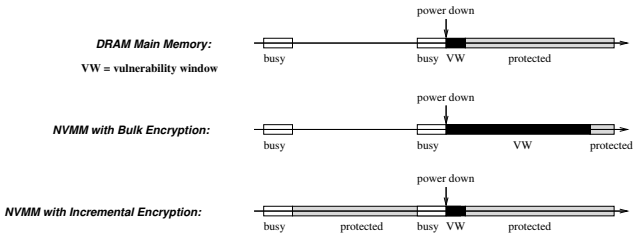


Figure 1: Contrasting the security protection in systems with DRAM main memory, NVMM with bulk encryption, and NVMM with incremental encryption.

window can be much longer than DRAM's retention time. In addition, with bulk encryption, the vulnerability window is a function of the size of the memory and will grow when larger main memory is provisioned in future systems. Another critical problem with such a long vulnerability window is that it may change the behavior of users who may perceive the encryption as an inconvenience or annoyance, and decide to bypass it, for example by powering down the system less frequently. This introduces new vulnerabilities.

Finally, the figure shows a case in which the system uses NVMM with incremental encryption, where different parts of memory are encrypted at different time so that, and at any given time, most of the memory is in encrypted form. Because only a small fraction of the memory needs to be encrypted at power down, the vulnerability window is much shorter, matching the DRAM's retention time. It must also be noted that with incremental encryption, the fraction of main memory to be encrypted is solely a function of the working set of the application(s) running when the system was powered down and does not depend on the size of the main memory and hence, unlike bulk encryption, the vulnerability window does not grow linearly with the size of the memory.

In addition to matching DRAM's vulnerability window, incremental encryption provides a *bonus protection* during the time the system is not powered down. Consider an attack scenario where the attacker steals a running system (for example, a user left the system locked and the system was stolen). Using bulk encryption only on power down, the entire memory will be visible to the attacker as long as the attacker can avoid powering down the system. With incremental encryption, there is a high probability that everything will be encrypted whenever the system is idle but is still powered. Note that such attacks are also possible in DRAM based systems, and we can avoid them by applying incremental encryption on DRAM as well.

4. DESIGNING INCREMENTAL ENCRYPTION

Recall that the desired solution to the attacks described in Section 3 must satisfy four requirements: (1) must retain the instant-on experience of an NVMM, (2) it must be self contained in the memory, (3) the time to encrypt the entire memory at power down must be in the order of a few seconds to match DRAM's retention time, and (4) it should have small performance and energy overheads.

The first requirement can clearly be satisfied through encrypting data in memory, rather than discarding or flushing it. The second requirement precludes solutions that require changes to a particular processor architecture and ISA. The requirement can be satisfied by architecting the solution entirely in the memory system, allowing the solution to be used in many processor systems in various platforms: servers, desktops, laptops, embedded systems, cell phones, etc. Having a self-contained solution means that the encryption engine must be located in the main memory module or device.

The third requirement is more challenging to deal with, and conflicts with the fourth requirement. On one extreme, bulk encryption, which encrypts the entire memory on power down, does not incur execution time overheads during regular execution, but cannot match DRAM's retention time (analysis in Section 7.3). In addition, as discussed in the previous section, it incurs an additional security concern due to the inconvenience it may cause to users. At another extreme, one can keep the entire memory encrypted at all time, which obviously has zero retention time at power down, but suffers from high performance overheads because every memory access by the processor must incur decryption latency, violating the fourth requirement.

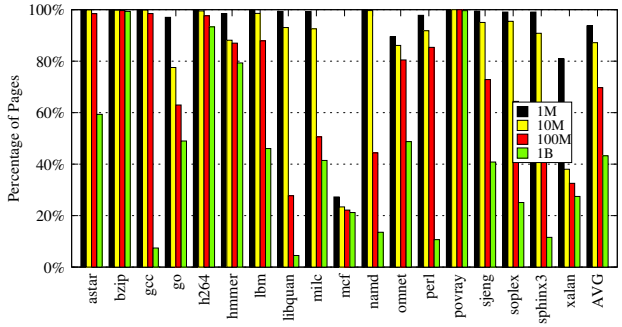
To satisfy both the third and fourth requirements, we can encrypt the majority of the main memory so that only a small remaining fraction of the main memory needs to be encrypted on power down. To minimize the performance overheads, we must be selective in what pages to encrypt. It is well known that the working set of an application (consisting of memory pages that an application is actively using) is much smaller than its resident set (consisting of all memory pages belonging to the application). By identifying the working set and encrypting the non-working part of the resident set, we can maximize the fraction of main memory that is encrypted but at the same time minimize the performance impact of memory encryption. Such an observation argues for *incremental* memory encryption, meaning that different data in the main memory is encrypted at different times depending on whether the data is predicted to still be useful to the processor. We refer to the pages that are not likely to be needed by the processor for a long time as *inert* pages. In the next few sections we will characterize inert pages to see how and how well we can predict them.

4.1 Inert Pages

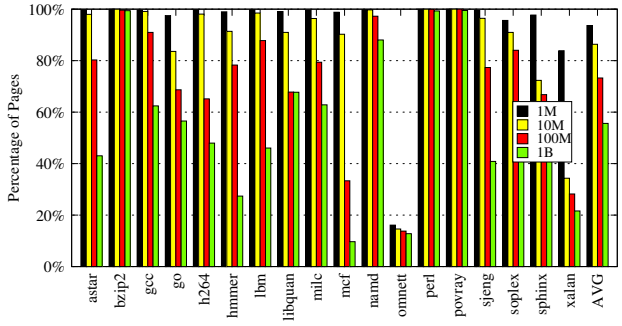
The main reason why there are inert pages is because of the temporal and spatial locality of memory accesses. Instructions and data recently accessed by the program are likely to be accessed again in the near future. This implies that other instructions and data are not going to be accessed by the processor in the near future, thus more likely to be inert. At the granularity of pages, even spatial locality of accesses within a page manifests as temporal reuse of the page. All these imply that the inter-access time will be very uneven, small for pages that are still in use by the application, but large or very large for (inert) pages that are no longer in use by the application. Such a phenomenon makes inert pages good candidates for encryption as they are unlikely to be a part of the *current* working set. Note that inert pages can be quite different from dead pages. Dead pages will never be accessed again in the future, but inert pages may be be accessed again but not in the near future.

Figure 2 shows the percentage of application pages that were not accessed by the application in the last 1M, 10M, 100M and 1B instructions leading to the end of simulation

period of 1B instructions after skipping 1B instructions (a) or 5B instructions (b). The skip 1B case represents an initial phase of application execution, while the skip 5B case represents more of the steady state execution. The figure shows the abundance of inert pages: on average, 43% of pages are never accessed in the execution window of the 1B simulation case. For the 5B case, this average increases to 56% (and further to 63% for 10B case – not shown in the figure).



(a) Skip1B Sim1B



(b) Skip5B Sim1B

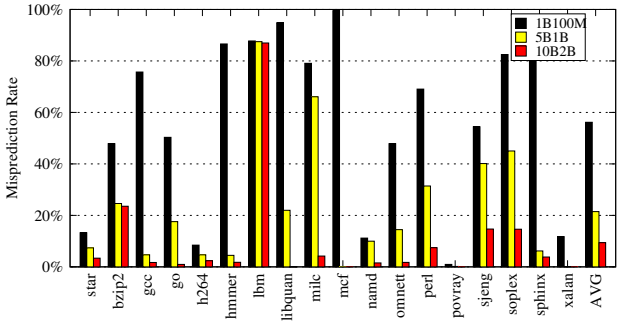
Figure 2: Inert pages for SPEC2006 benchmarks.

4.2 Inert Page Prediction

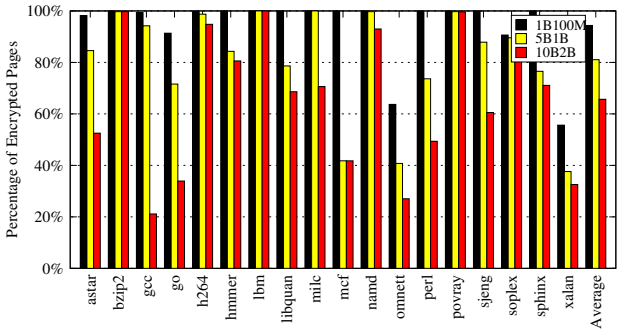
To exploit inert pages, our incremental encryption i-NVMM needs to predict them, using a structure we refer to as *Inert Page Predictor (IPP)*. Before we discuss how we implement the IPP, it is important to consider what metrics can evaluate the goodness of a predictor. One important metric is **coverage**, defined as the percentage of pages in memory that are encrypted at the time of a power event (at the end of simulation period for our experiments). The higher the coverage, the less time is needed to encrypt the remaining pages at power down. Another important metric is **misprediction rate**, defined as the percentage of accesses that are to predicted-inert pages. Misprediction is harmful because it incurs page decryption latency when a processor accesses a page that is currently encrypted, and because it may incur additional writes to the NVMM that reduces the write endurance of NVMM. The last important metric is **performance overhead** as a result of decryption latencies during regular execution. Ideally, an IPP design should have high coverage, low misprediction rate and low performance overhead.

4.2.1 When to encrypt?

One decision our IPP must make is when to predict a page has become inert and can be encrypted. IPP cannot rely on a complex predictor structure since it will be implemented at the memory side where it must meet a tight power envelope. Taking this constraint into account, we investigate if we can rely on a simple input that tracks whether a page has not been accessed for a while. To achieve that, we define *Scan interval (SI)* as a periodic interval at which the memory is scanned to identify inert pages. We also define *Predict-Inert Threshold (PIT)* as the length of inactivity period after which a page is predicted as inert. The values of SI and PIT must be chosen carefully. If they are too small, pages may be predicted inert while the application is using them, resulting in high coverage but high misprediction rate and performance overheads. On the other hand, too large values for SI and PIT will be more conservative, resulting in lower misprediction rate and performance overheads, but may result in a poor coverage. Figure 3 shows the misprediction rate and coverage for varying SI and PIT values.



(a) Misprediction Rate



(b) Coverage

Figure 3: Impact of different values of the scan interval (SI) and predict inert threshold (PIT).

As can be seen from the figure, using a small SI of 1B cycles and a small PIT of 100M cycles, a high coverage of 94% is achieved but at the same time, it results in a high misprediction rate of 56% on an average. On the other hand, using a very large SI of 10B cycles and a large PIT of 2B cycles results in a low misprediction rate of 9.3% on an average but at the same time, it results in a low coverage averaging at 65%. Using a moderate SI of 5B cycles and a moderate PIT of 1B cycles achieves a good compromise, a relatively low misprediction rate of 21% and a relatively high coverage of

81% on an average. Based on this analysis, we fix the SI at 5B cycles and PIT at 1B cycles for the IPP and use these values for the rest of our analysis.

4.2.2 When to decrypt?

Another important design issue for IPP is its behavior on a misprediction, which occurs when there is an access to a page that is in encrypted form (because it was predicted inert a while back). Exposed decryption latency directly impacts the performance overheads. To keep the overheads low, one option is to always decrypt a page on the first access, to avoid decryption latencies for future accesses to the same page. Such an option will lower performance overheads but may over-eagerly decrypt pages that are only accessed a few times, incurring several adverse effects: decreased coverage, increased writes to the main memory (due to writing back the newly decrypted pages), and hence reduced write endurance. At another extreme, we can keep encrypted page encrypted, but instead only decrypt the accessed block and return it to the processor.

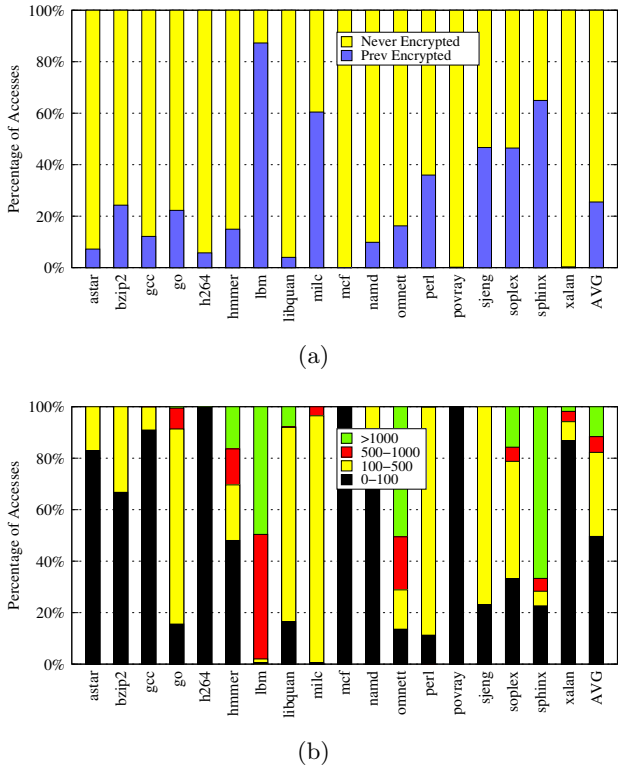


Figure 4: Distribution of accesses to previously predicted-inert (encrypted) pages vs. never encrypted pages (a), and the distribution of accesses to only previously predicted-inert pages (b).

To discover what policy is good to use, we plot the distribution of accesses to pages that have been predicted inert in the past, versus accesses to pages that have not been predicted inert (Figure 4(a)). We refer to previously predicted-inert pages as encrypted pages, although for this characterization, we do not actually encrypt the pages. The figure shows that a significant fraction of accesses are to pages that were never encrypted (on average 75% for the first five billion instructions simulated). However, for some applications (e.g. lbm, milc, sjeng, soplex, sphinx), more than 40% of all accesses are to previously encrypted pages, meaning that more than 40% of the accesses will suffer a decryption latency which is directly in the critical path of the processor performance. In addition, never decrypting a page on a misprediction results in an eventual convergence to the entire memory being encrypted, which worsens the performance overheads over time. Therefore, there needs to be a policy where a page is decrypted after n-accesses, where the *page decryption threshold* (n) is an adjustable parameter.

In order to discover how many accesses to a page should trigger decryption, we characterize the statistical distribution of the number of accesses that occur to encrypted pages. Figure 4(b) shows the number of accesses occurring to encrypted pages for the first 5 billion instructions simulated, divided into bins of 0 to 100 accesses, 100 to 500 accesses, 500 to 1000 accesses and greater than 1000 accesses. On average, roughly 50% of pages are accessed more than 100 times after they are predicted as inert. However, for applications which showed a large percentage ($> 40\%$) of accesses to encrypted pages (lbm, milc, sjeng, soplex, sphinx), a much larger percentage of pages ($> 80\%$ on average, and at least 64%) are accessed more than 100 times. Thus, a relatively large page decryption threshold (e.g. hundreds of accesses per page) can avoid over-actively decrypting pages for applications that do not access encrypted pages much, and at the same time allow a page to be decrypted early enough to avoid many more subsequent accesses from suffering from decryption latency.

For further insights, we analyze two of the benchmarks *lbm and sphinx*'s distribution of number of accesses to each of their encrypted pages. Each point in Figure 5 shows the page number (x-axes) and the number of accesses the page receives (y-axes). Figure 5(a) shows that *lbm*'s predicted-inert pages are accessed roughly 1000 times on average. Figure 5(b) also shows that majority of *sphinx*'s predicted-inert pages are accessed more than 1000 times, and up to 25K times. This indicates that pages that were previously predicted inert do not stay inert for the lifetime of the application, and therefore page decryption needs to be triggered at some point. In addition, a relatively large threshold (e.g. tens to hundreds of accesses) for triggering page decryption is acceptable given that the pages are accessed for thousands of times.

For other benchmarks, such as *gcc and perl* (not shown in the figure), nearly all the pages predicted inert receive less than 500 accesses. However, they have outliers where some predicted-inert pages receive a large number of accesses (250K for *gcc* and 600K for *perl*). Hence, even for these benchmarks, predicted-inert pages can benefit from page decryption being triggered after some threshold.

Overall, it is clear that across a broad range of applications, it is beneficial to trigger page decryption for predicted-inert pages, using a reasonable threshold of accesses (likely in the order of hundreds of accesses). In Section 7, we will experiment with how page decryption threshold values affect performance.

4.2.3 Hiding Decryption Latency Overheads

We have discussed what criteria can be used to predict a page as inert and encrypt it, and what criteria can be used to decrypt it. However, with the decryption threshold, for each page that is predicted inert and encrypted, there are

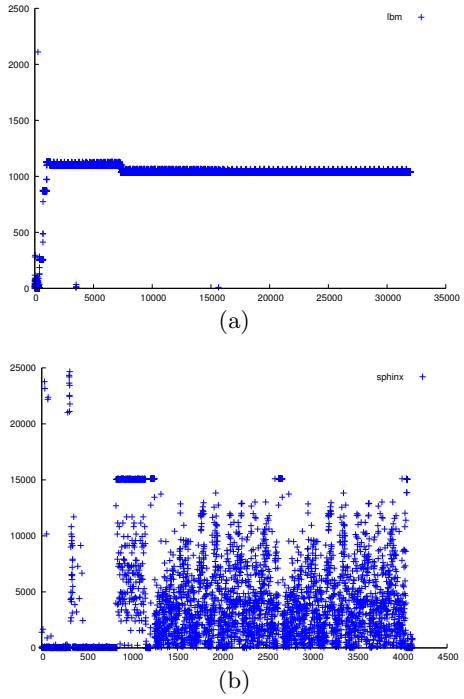


Figure 5: Distribution of accesses to previously encrypted (inert) pages. Each point in the graph represents an inert page with the y-axis value representing the number of accesses to that inert page.

still quite a few accesses that must suffer from decryption latency. Thus, we further explore a *pre-decryption mechanism*, where we predict a page that will be accessed soon in the future and pre-decrypt it before it is accessed by the processor. We borrow techniques from Markov prefetching to achieve this [6] to design a *correlation pre-decryption* mechanism.

Correlation pre-decryption uses page address correlation information as input to predict the next page that will be accessed, and pre-decrypt it ahead of time. For each physical page, a *NextPage* field is maintained (Section 5), which stores the page number that, in the most recent history, was accessed next after this page. When a page in encrypted state is accessed, it is decrypted and its *next page* entry is looked up. If the physical page located at the *next page* entry is also in encrypted state, it is decrypted as well. Note that this mechanism relies on physical page addresses, so it may happen that the next page is no longer in memory, and in its place is a different page. In such a case, pre-decrypting the page results in wasted work, but does not impact correctness.

5. ARCHITECTURE SUPPORT FOR INCREMENTAL ENCRYPTION

In order to perform inert page prediction, i-NVMM keeps track of the status of each page and various parameters that are common to all pages. We choose a page size of 4KB, which is not too large to cause non-negligible storage overheads for keeping a few of them buffered at the cryptographic engine and to cause very high encryption/decryption latencies. It is also not too small that the cost of tracking pages becomes too high. In addition, Operating System page sizes are often 4KB as well in most systems. Many applications and Operating System codes have been tuned to maximize spatial and temporal locality for such page size, and hence choosing our page size to match the OS page size or its multiple has a merit of enjoying better spatial and temporal locality compared to other sizes.

Figure 6 shows hardware components that are added to a non-volatile main memory (NVMM) module [2]. One added component is *Page Status Table* (PST), an SRAM structure that keeps track of the status of each page: 1-bit that indicates whether a page is currently encrypted or not (EncStatus), the last time the page was accessed (LastAcc), the number of times the page has been accessed (numAcc), the next page accessed after this one (NextPage), and 1-bit that indicates whether a page is pending for encryption/decryption or not (Pending). LastAcc is used by the Inert Page Prediction (IPP) to predict a page as inert in order to encrypt it. numAcc is used for tracking how many accesses have been received by a currently-encrypted page in order to decide when it should be queued for decryption.

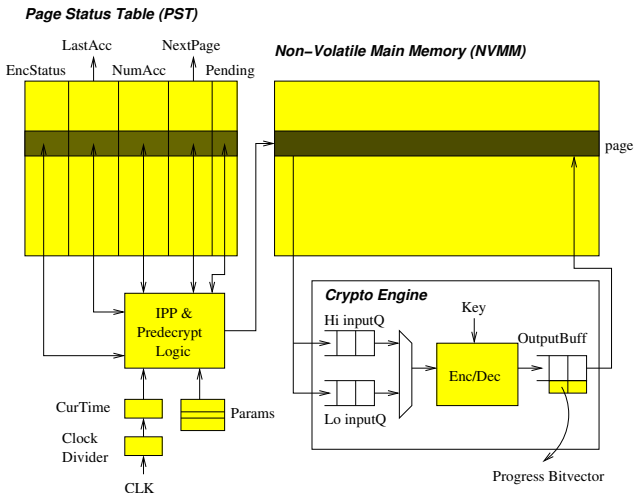


Figure 6: High-level diagram of i-NVMM architecture support.

Assuming a 32GB PCM module and a 4KB page, NextPage will require 23 bits of storage. numAcc requires 9 bits if the page decryption threshold is 500. The storage requirement for LastAcc depends on how fine-grained time is tracked. If it is incremented every 1 million clock cycles, then a 44-bit field (which can track 2^{64} cycles) is large enough to last a millennia. Overall, for each page, the metadata storage requirement is $23 + 9 + 44 + 1 + 1 = 78$ bits, or 0.2% of the total memory size.

Next, i-NVMM needs to predict whether a non-encrypted page becomes inert, and predict the next page that will be accessed after the current one in order to support pre-decryption. This is achieved by adding a logic structure shown as *IPP & Predecrypt Logic*. The logic needs to keep

[2]For our discussion, we have assumed non-volatile memory alone to form the main memory. However, our mechanisms can be adapted in a straightforward way to hybrid volatile/non-volatile main memory systems such as the one proposed in [16].

track of time since the last access for multiple pages. It receives time signal from a clock divider, which divides the clock signal of the module into a time interval that is a lot coarser, for example, a time signal every 1 million clock ticks or more. Every time interval, the logic increments a current time field (CurTime) that it maintains to reflect the current time interval. Every time a page is accessed, the logic updates its LastAcc value to the current value of the time maintained by the logic. At every multiples of the time interval, corresponding to the scan interval (SI) in the parameter table, the LastAcc value of each page that is not currently encrypted is checked against the page inert threshold (PIT). If the difference between the CurTime and LastAcc field is larger than the PIT, it indicates that enough time has elapsed that the page can be predicted as inert. The page is then sent to an input queue in the cryptographic engine for encryption.

In addition to storing the SI and PIT parameters for the IPP, the parameter table also stores the page decryption threshold (PDT) value, which determines how many accesses should occur to an encrypted page before the page is queued for decryption. The parameters at the parameter table can be programmed by the NVMM vendor at the installation of the system. If desired, the parameters can be made configurable through BIOS settings, as long as they are protected by password.

The cryptographic engine in the memory module has two input buffers with different priorities (high and low). When a page is predicted inert by IPP, the page is queued for encryption at the low priority input queue (Lo inputQ). If a block or page is demand-fetched by the processor and needs to be decrypted, the block or page is queued at the high priority input queue (Hi inputQ) instead. The cryptographic engine uses the secret key and cryptographic-strength encryption algorithm such as AES to encrypt or decrypt a page or block. It serves requests from the high priority and empties it before it serves requests from the low priority queue, in order to avoid critical-path delay for processor requests. In the rare case in which predicted-inert pages that need to be encrypted cannot be inserted to the low priority queue because it is full, the page can be tagged with a flag (Pending bit is set) that indicates it is pending for encryption. In the next scan interval, the IPP retries inserting the request to encrypt the page into the low priority queue.

There are four buffers in the cryptographic engine's output. The first output buffer stores the most recently encrypted page from the low priority input queue. Rather than directly writing it to the NVMM, we use a lazy approach by writing it back to the NVMM only when it needs to be evicted to make room for a new encrypted page. The reason for this is that if the inert prediction is wrong and the encrypted page (in the output buffer) is accessed by the processor, it can be discarded and we avoid writing to the NVMM. The second output buffer is used to store the decrypted page from the high priority input queue. The cryptographic engine is also equipped with two buffers for holding pre-decrypted pages. These buffers are used to prevent writing the decrypted output back to the main memory because pre-decryption is a speculative mechanism, and when the speculation is wrong, having these buffers helps prevent affecting the write endurance of the NVMM. A predecrypted page is written back to the main memory only if the prediction is confirmed by accesses to the page while the page resides in one of the two buffers. Otherwise, it is discarded. Each of the decryption output buffers also contains a frame number and a decryption progress bitvector. The frame number stores the physical location of the page being decrypted, while the progress bitvector tracks which blocks have been decrypted and which ones have not yet been decrypted. Each bit in the bitvector corresponds to a block in the page being decrypted, and is set when the block decryption is complete.

If there is a demand access from the processor to a page that resides in one of the output buffers (i.e. the page was just decrypted or is being decrypted), the progress bitvector is checked to see if the requested block has been decrypted. If it has, the block is returned to the processor. Otherwise, we employ *critical block first* policy by immediately queueing the block for decryption.

A crucial security decision regarding the design of i-NVMM is the secret key used for the cryptographic engine. One can embed the key in the memory module itself, but this weakens the security protection as an attacker with physical access to the system can scan the key off the memory module. Another option is to have the processor store the key on the processor chip, which is used by the memory module and is scrubbed off once the main memory has been fully encrypted. This solution, however, requires nonvolatile key storage at the processor, breaking the requirement of i-NVMM to be self-contained. A possible solution that provides both security and self-containment is to use two-factor authentication, using a secret key that is generated based on some form of external input, for example password typed by the user, or downloaded from a removable media (smart card or Near Field Communication (NFC) card), or secure network. An attacker cannot produce valid external input, hence a wrong decryption key will be generated. Details of key management protocols are not the focus of this paper. Interested readers are referred to [5].

One final security issue is handling an event where the attacker removes the power supply to the memory in order to disrupt full encryption. In order to deal with this attack, an internal CMOS battery can be provided to provide reserve power to complete encryption when the main power supply is disrupted. The cost of such a battery again favors keeping the vulnerability window as small as possible.

6. ENVIRONMENTAL SETUP

We evaluate i-NVMM on a Simics [11]-based full-system simulator. We model a 4GHz, in-order processor with split L1 data and instruction caches. Both caches are 32KB in size, 2-way set associative and have a 2-cycle round-trip hit latency. The L2 cache is unified and is 1MB in size, 8-way set associative, and has a 10-cycle round-trip hit latency. All caches have a 64B block size and use LRU replacement policy. Simics does not simulate paging functionality (both in terms of functionality and in timing). Thus, we add a virtual memory simulator to model the paging functionality of the OS, so that our memory module receives realistic physical address streams in order to enable the simulation of page loading and replacement. We use Phase-Change memory (PCM) as an example of the non-volatile main memory. PCM has a base read access latency of 1280 cycles on a 4GHz processor [16]. With various techniques for latency reduction and hiding [3, 9, 15, 16], it is reasonable to assume a smaller base read access latency of 960 cycles.

Highly optimized hardware implementation of various cryp-

tographic algorithms have been demonstrated in [7], where a 16-stage AES engine was shown to have an FO4 delay of 36ns and area of 5.3mm^2 with 1GHz clock. Taking into account that the memory system uses a lower clock frequency (1066MHz vs. 4GHz), a self-contained implementation on a memory module may have a latency 4–6× higher, measured in the processor clock frequency. Furthermore, the cryptographic engine in the memory may be optimized for lower power consumption and lower die area overhead compared to an on-processor-chip cryptographic engine, hence we further increase the AES encryption latency by a factor of 3–5×. Based on these conservative assumptions, AES latency is assumed to be 640 cycles, 18× of [7]. The contention at the cryptographic engine is modeled accurately. While weaker and faster encryption algorithms can be used, we show that even with AES with conservative latency estimates, our mechanisms can still achieve low overheads.

For the IPP parameters, we assume a scan interval (SI) of 5 billion cycles, a predict inert threshold (PIT) of 1 billion cycles (as established in Section 4), and a page decryption threshold (PDT) of 500 (i.e., a currently-encrypted page is decrypted after 500 accesses). We use all (but one) C/C++ SPEC2006 benchmarks. Only one benchmark, dealII, is excluded due to a compilation problem. For each simulation, we use the reference input set and simulate for first 5 billion instructions.

7. EVALUATION

7.1 Incremental vs. Full Encryption

In our first evaluation, we compare the performance overheads of our incremental encryption with a base scheme which keeps the main memory encrypted at all times. The latter scheme eliminates the security vulnerability of lingering data in an NVMM, but each time a processor accesses a block, it will suffer a full decryption latency for the block. Figure 7 compares the execution time overheads and coverage of this base scheme versus our incremental encryption utilizing just inert page prediction for encrypting the memory (ippEnc), and adding pre-decryption on top of that (ippEnc+PreDec).

Figure 7(a) shows that the performance overheads of full memory encryption are unacceptably high, averaging 42% and exceeding 60% for four benchmarks. In comparison, incremental encryption (ippEnc) achieves an average overhead of only 7.2%, with overheads lower than 20% for all but two benchmarks. Adding pre-decryption (ippEnc+PreDec) lowers the overheads further to 3.7%, with overheads lower than 20% for all but one benchmark. The average overhead of ippEnc+PreDec is roughly one order of magnitude lower than full encryption. For a hypothetical scheme that randomly encrypt pages to achieve 3.7% overheads, it can only keep $\frac{3.7}{42} \times 100 = 8.8\%$ all pages encrypted. However, by encrypting only inert (rather than random) pages, Figure 7(b) shows that ippEnc and ippEnc+PreDec keep 78% of all pages encrypted at all time on average. It should be noted that predecryption does not impact coverage directly, but may indirectly affect coverage slightly, due to changing the timing of accesses made by the application, as seen in Figure 7(b).

Even for benchmarks (e.g. *bzip2*) with nearly 100% coverage, our scheme maintains outperforms full encryption. The reason is that in full encryption, each memory access suffers a full decryption delay, whereas in incremental encryption, only accesses to already-encrypted pages suffer decryption delay. Such difference is crucial for applications with streaming memory access behavior. These streaming applications at first allocate a page in plaintext form, incurring no decryption delay, and the page is encrypted only after it is unused for a while. For example, benchmarks like *bzip2* keep the active pages not-encrypted due to its streaming spatial reuse, but keep most (99.7%) other pages encrypted due to the lack of temporal reuse, thereby achieving low performance overheads even with a near 100% coverage. These results argue strongly for the merit of incremental encryption over full encryption.

As shown in Figure 7(b), i-NVMM yields variable coverage across benchmarks. This leads us to an interesting alternative where we can enforce a constant target coverage to keep a constant vulnerability window, which may be useful for very limited-battery devices, at the expense of less predictable performance overheads. However, since our focus is on low performance overheads, we do not investigate this alternative further.

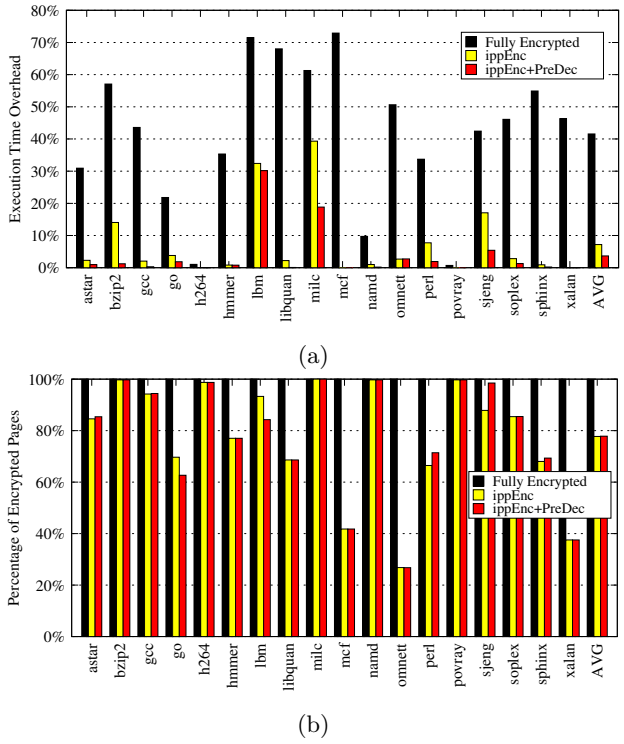


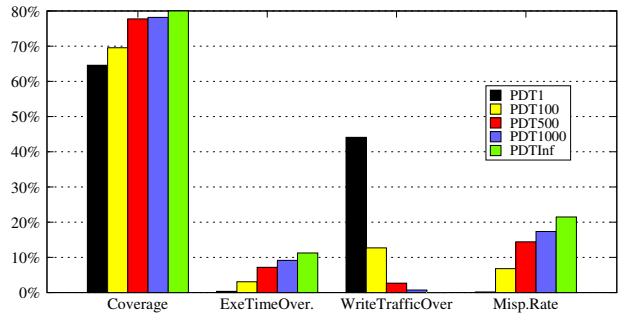
Figure 7: Comparing incremental encryption versus a fully-encrypting the main memory: execution time overheads (a), and percent of all pages that are kept in encrypted form (b).

7.2 Misprediction Handling Policies

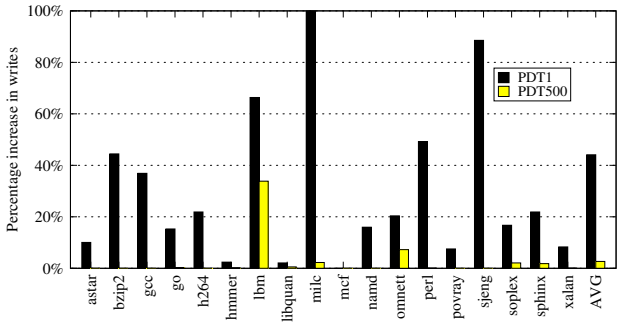
In i-NVMM, we have three policy choices for handling *misprediction*. The first policy is to decrypt a page on the first access to the page. At the other extreme, we can choose not to decrypt the page at all, but instead decrypt only the block accessed by the processor. Finally, we can decrypt a page after a certain number of accesses (defined by the page

decryption threshold) has elapsed. These policies can be thought of as one policy relying on different page decryption thresholds: PDT1 for decryption on first access, PDTx for decryption after x accesses, and PDT∞ for never decrypting an encrypted page. Note that PDT500 is our default policy (used in the previous section).

Figure 8(a) compares the coverage (percent of pages encrypted at the end of simulation), execution time overheads, the increase in the number of writes compared to a system without security protection and the misprediction rate. The figure shows that as a higher page decryption threshold is used, coverage also increases. However, since coverage improvement slows down significantly from PDT of 500 and higher, it indicates that there is little coverage benefit from employing PDT higher than 500.



(a)



(b)

Figure 8: Comparing PDT1, PDT100, PDT500, PDT1000, PDT∞ (a), increase in writes to memory for PDT1 vs PDT500 (b).

In terms of execution time, we can see that using a lower PDT results in lower execution time overheads, because as pages are decrypted earlier, more accesses to the page do not suffer from decryption latencies. The average overheads for PDT1, PDT100, PDT500, PDT1000, and PDT∞ are 0.4%, 3.1%, 7.2%, 9.1%, and 11.3%. This implies if everything else is equal, a lower PDT should be preferred. Unfortunately, a lower PDT also incurs a lower coverage, which means that the vulnerability window may be too high that it compromises security. For example, PDT1's vulnerability window is roughly twice as much as PDT500's vulnerability window (Section 7.3).

Furthermore, a low PDT incurs more additional writes to the NVMM, because an eagerly-decrypted page must be written back to the NVMM, decreasing write endurance. The figure shows that the number of writes is increased by 44% on average with PDT1, versus only 2.7% for PDT500. The reason for such a high percentage increase in writes is because there are a lot more read than write memory accesses, and some of these reads now incur a page decryption write back. Hence, even a small increase in page decryption can result in a large percentage increase in writes to the main memory. While it may seem that 44% average increase in writes is tolerable (31% ($1 - \frac{100}{144}$) decrease in write endurance), there are several benchmarks for which the number of writes increases by much more than the average. Figure 8(b) compares the increase in writes for PDT1 versus PDT500. The increase in writes for three benchmarks exceed 60% and in one benchmark (*milc*), the increase is 366%, corresponding to 79% ($1 - \frac{100}{466}$) reduction in write endurance. Clearly, a low PDT value such as PDT1 results in an unacceptable decline in write endurance. Overall, we can conclude that PDT value of 500 gives a good combination of low execution time overhead, high coverage, and acceptable write endurance. Finally, the figure shows an increasing misprediction rate as we use a higher PDT because by decrypting pages more lazily, there are more accesses to encrypted pages.

7.3 i-NVMM Vulnerability Window

As discussed in Section 3, our incremental encryption is more secure compared to bulk encryption due to a shorter vulnerability window, and that much of the main memory is encrypted at all time, not just after power down. To estimate the vulnerability window, we calculate the time required to fully encrypt the memory at power-down. For each page, it will take $640 + (40 * 255) = 10840$ cycles (AES operates on 16-byte blocks and after the first block, each block will take 40-cycles assuming a 16-stage pipelined engine). We also assume that reading a page can be overlapped with the encryption latency of the previous page. Hence, for a 32GB memory, the vulnerability window for bulk encryption can be calculated as: $\frac{32GB}{4KB} * 10840$ cycles (23 seconds), as it has to encrypt the entire memory at power down. For incremental encryption, the vulnerability window will be decided based on the coverage. We quantify and compare the vulnerability window of incremental encryption in our scheme with bulk encryption, and show the results in Figure 9.

The vulnerability window decreases to an average of 5 seconds when incremental encryption is used. There are only three benchmarks in which the vulnerability window exceeds 10 seconds. In addition to reducing the vulnerability window significantly, recall that incremental encryption provides additional benefits. First, incremental encryption provides security protection *not only* at power down, but also anytime the system is idle for a few seconds, the entire memory becomes encrypted. Bulk encryption does not provide such a benefit, because if attackers steal a system that is still powered on (quite likely with mobile devices), the main memory is completely exposed. Secondly, a shorter vulnerability window in incremental encryption requires $\tilde{8}0\%$ less reserve battery power than bulk encryption to complete encryption at power down. Thirdly, the vulnerability window of bulk encryption scales with the memory size, whereas i-NVMM's

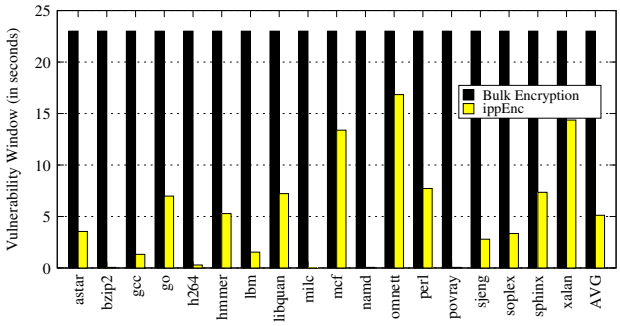


Figure 9: Vulnerability Window for i-NVMM vs bulk encryption.

vulnerability window scales only with active applications' working set size. In a real system, it is likely that only a small number of applications are actively running, while the main memory contains pages of all (active and blocked) applications. Hence, the fraction of pages that that are not yet encrypted may be much smaller than our estimate. Hence, our results presented here show a pessimistic upper-bound of the vulnerability window of i-NVMM. Finally, a critical danger of bulk encryption's long vulnerability window is that it may change the behavior of users who may perceive the encryption as an inconvenience or annoyance, and decide to bypass it, for example by powering down the system less frequently. This introduces new security vulnerabilities.

7.4 Other Results

Effectiveness of Pre-decryption. In Figure 7(a), we have shown that the execution time overheads can be reduced significantly when we couple inert page prediction with correlation pre-decryption. Figure 10 shows percentage of accesses to encrypted pages that are already fully pre-decrypted ("Totally satisfied") or are being pre-decrypted ("Partially satisfied"). The figure explains why correlation pre-decryption is effective: on average 55% of accesses to encrypted pages do not suffer from a full decryption latency. Considering that the pre-decryption mechanism comes with little hardware cost (22-bit NextPage field in the Page Status Table, and two 4KB SRAM buffers to hold pre-decrypted pages at the output of the cryptographic engine), it is a cost-effective technique to employ.

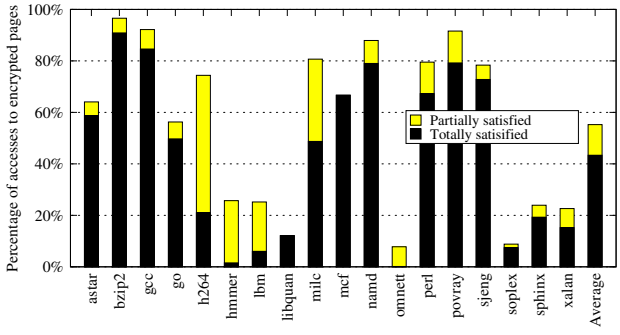


Figure 10: Effectiveness of Pre-decryption.

Energy Overheads of i-NVMM. i-NVMM incurs additional energy consumption for cryptographic operations compared to a main memory that has no security protection. To obtain the energy overhead estimates for i-NVMM, we use cryptographic energy numbers from [12], DRAM energy numbers obtained using DDR3 power calculator [13], adjusted for PCM read and write energy assuming that PCM's read and write energy are 4× and 43× those of DRAM. These estimates are shown in Figure 11. Overall, i-NVMM adds an energy overhead of 5.1% on average. This is an attractive figure considering the additional security protection provided by our scheme.

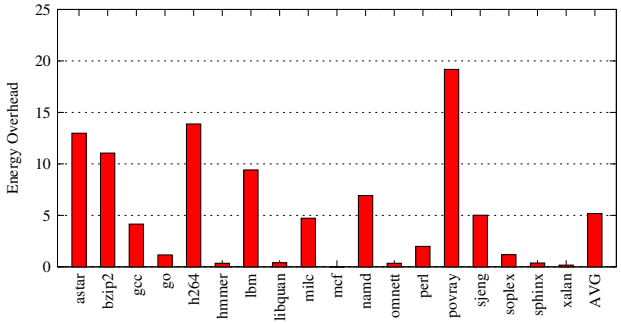


Figure 11: Energy Overheads.

Sensitivity to AES Latency. Recall that our AES latency assumption of 640 cycles is quite conservative, at 18× an optimized version implementable in a processor chip. In Figure 12, we explore the execution time overheads if the AES latency is less conservative (320 cycles), or very conservative (960 cycles). The execution time overheads are reduced by more than a half with a 320-cycle latency (1.6% on average vs. 3.7%), and increases to 5.5% with a 960-cycle latency. Hence, a faster or more optimized engine will definitely reduce the performance overheads of i-NVMM further, while a less sophisticated engine may still provide a reasonable overhead.

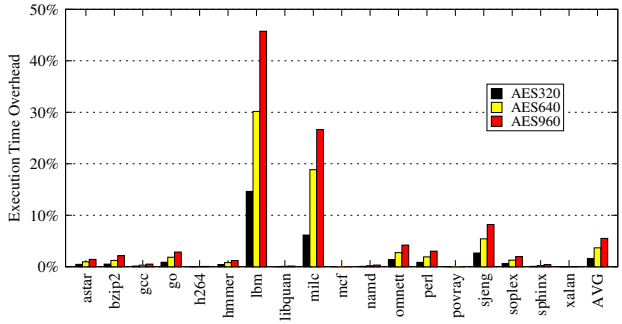


Figure 12: Impact of AES latency.

8. CONCLUSIONS

A non-volatile main memory system poses a critical security threat to the privacy of the application data in main memory as the memory retains data even when the system is powered down. In this paper, we have proposed our solution for the problem (i-NVMM) using an incremental encryption approach. i-NVMM satisfies all four requirements

for a satisfactory solution. First, by retaining data in the main memory (in encrypted form), the instant-on experience of non-volatile main memory is preserved. Second, the architecture of i-NVMM is self contained in the memory module, allowing i-NVMM to be a high-volume commodity memory product that can be used in a wide range of processor platforms it is attached to, independent of specific ISA and specific support in the processor architecture. Third, on average i-NVMM can encrypt the entire main memory in 5 seconds, matching DRAM's retention time after power down, because i-NVMM keeps 78% of the main memory encrypted and hence only a small remaining fraction must be encrypted at power down. Finally, i-NVMM uses relatively simple hardware support that incurs only a small execution time overhead of 3.7% on average, energy overhead of 5.1%, and has a negligible impact on the write endurance.

9. REFERENCES

[1] *International Technology Roadmap for Semiconductors, ITRS 2007.*

[2] S. Chhabra and Y. Solihin. Defining Anomalous Behavior for Phase Change Memory. *Workshop on the Use of Emerging Storage and Memory Technologies*, held in conjunction with HPCA, 2010.

[3] S. Cho and H. Lee. Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance. In *Proc. of the 42nd Annual International Symposium on Microarchitecture"*, 2009.

[4] T. Gilmont, J.-D. Legat, and J.-J. Quisquater. Enhancing the Security in the Memory Management Unit. In *Proc. of the 25th EuroMicro Conference*, 1999.

[5] J. Halderman, S. Schoen, N. Heninger, W. Clarkson, W. Paul, J. Calandrino, A. Feldman, J. Appelbaum, and E. Felten. Lest We Remember: Cold Boot Attacks on Encryption Keys. In *USENIX Security Symposium*, 2008.

[6] D. Joseph and D. Grunwald. Prefetching Using Markov Predictors. In *ISCA*, 1997.

[7] T. Kgil, L. Falk, and T. Mudge. ChipLock: Support for Secure Microarchitectures. In *Proceedings of the Workshop on Architectural Support for Security and Anti-Virus (WASSA)*, Oct. 2004.

[8] J. Kong and H. Zhou. Improving privacy and lifetime of pcm-based main memory. In *DSN' 10*. IEEE, 2010.

[9] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th International Symposium on Computer architecture*, pages 2–13, New York, NY, USA, 2009. ACM.

[10] C. Lefurgy et al. Energy management for commercial servers. *IEEE Computer*, 36(12):39–48, 2003.

[11] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A Full System Simulation Platform. *IEEE Computer Society*, 35(2):50–58, 2002.

[12] S. Mathew et al. A 2.4-53Gbps On-die AES/GFMUL/SHA/TRNG Multi-protocol Encryption Engine for High-performance Microprocessors in 45nm CMOS . In *ISSCC*, 2010.

[13] Micron. Micron Sytem Power Calculator. *download.micron.com/downloads/misc/ddr3_power_calc.xls*.

[14] M. K. Qureshi. et al. Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. In *Proc. of the 42nd Annual International Symposium on Microarchitecture"*, 2009.

[15] M. K. Qureshi, M. M. Franceschini, L. A. Lastras-Monta no, and J. P. Karidis. Morphable memory system: a robust architecture for exploiting multi-level phase change memories. In *ISCA '10: Proceedings of the 37th annual international symposium on Computer architecture*, pages 153–162. ACM, 2010.

[16] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th International Symposium on Computer architecture*, pages 24–33. ACM, 2009.

[17] S. Raoux. et al. Phase-change random access memory: a scalable technology. *IBM J. Res. Dev.*, 52(4):465–479, 2008.

[18] P. Roberts. MIT: Discarded hard drives yield private info. *ComputerWorld*, Jan 16, 2003.

[19] B. Rogers, S. Chhabra, Y. Solihin, and M. Prvulovic. Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly. In *MICRO*, 2007.

[20] B. Rogers, Y. Solihin, and M. Prvulovic. Efficient data protection for distributed shared memory multiprocessors. In *International Conference on Parallel Architectures and Compilation Techniques*, 2006.

[21] U. Ruoss, D. Ielmini, and A. Lacaita. Analytical Modeling of Chalcogenide Crystallization for PCM Data-Retention Extrapolation. *IEEE Transactions on Electron Devices*, 54(10), 2007.

[22] G. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. Efficient Memory Integrity Verification and Encryption for Secure Processor. In *Proc. of the 36th Annual International Symposium on Microarchitecture*, 2003.

[23] R. Venkatesan, S. Herr, and E. Rotenberg. Retention-aware placement in dram (rapid): Software methods for quasi-non-volatile dram. In *HPCA*, pages 157–167. IEEE, 2006.

[24] W. Zhang and T. Li. Exploring phase change memory and 3d die-stacking for power/thermal friendly, fast and durable memory architectures. In *PACT '09: Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques*, pages 101–112. IEEE Computer Society, 2009.

[25] W. Zhang and T. Li. Exploring phase change memory and 3d die-stacking for power/thermal friendly, fast and durable memory architectures. In *Proc of the 16th International Conference on Parallel Architectures and Compilation Techniques (PACT-18)*, 2009.

[26] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A durable and energy efficient main memory using phase change memory technology. In *Proceedings of the 36th annual International Symposium on Computer architecture*, pages 14–23. ACM, 2009.